

JTRS Have Quick Physical API Service Definition

V1.0
December 15, 2000

Prepared for the
Joint Tactical Radio System (JTRS) Joint Program Office

Prepared by the
Modular Software-programmable Radio Consortium
Under Contract No. DAAB15-00-3-0001

Revision Summary

1.0	Initial release

Table of Contents

1	INTRODUCTION.....	1
1.1	OVERVIEW.....	1
1.2	SERVICE LAYER DESCRIPTION.....	2
1.2.1	Non-Real-Time Services.....	2
1.2.2	Real-Time Services.....	2
1.3	MODES OF SERVICE.....	2
1.4	SERVICE STATES.....	2
1.5	REFERENCED DOCUMENTS.....	2
2	UUID.....	3
3	SERVICES.....	3
3.1	NON-REAL-TIME SERVICES.....	4
3.1.1	TransceiverSetup.....	4
3.1.2	RadioMode.....	4
3.1.3	MediaSetup.....	6
3.1.4	ReceiveCommand.....	6
3.2	REAL-TIME SERVICES.....	6
3.2.1	UlongPacket.....	7
3.2.2	UshortPacket.....	8
3.2.3	OctetPacket.....	9
4	SERVICE PRIMITIVES.....	10
4.1	TRANSCEIVER SETUP.....	10
4.1.1	setUpReceiverParams.....	10
4.1.2	setUpTransmitterParams.....	11
4.1.3	setDate.....	13
4.2	RADIOMODE.....	14
4.2.1	setRadioMode.....	14
4.3	MEDIASETUP.....	15
4.4	ULONGPACKET.....	16
4.4.1	pushPacket.....	16
4.4.2	spaceAvailable.....	17
4.4.3	enableFlowControlSignals.....	18
4.4.4	enableEmptySignal.....	18
4.4.5	setNumOfPriorityQueues.....	19
4.5	USHORTPACKET	20
4.5.1	pushPacket.....	20
4.5.2	spaceAvailable.....	21
4.5.3	enableFlowControlSignals.....	21
4.5.4	enableEmptySignal.....	22
4.5.5	setNumOfPriorityQueues.....	22
4.6	OCTETPACKET	23
4.6.1	pushPacket.....	23
4.6.2	spaceAvailable.....	24
4.6.3	enableFlowControlSignals.....	25

4.6.4	enableEmptySignal.....	25
4.6.5	setNumOfPriorityQueues.....	26
4.7	RECEIVECOMMAND.....	27
4.7.1	Receive.....	27
5	ALLOWABLE SEQUENCE OF SERVICE PRIMITIVES.....	28
6	UTILIZATION OF PHYSICAL NON-REAL-TIME BUILDING BLOCKS..	29
7	PRECEDENCE OF SERVICE PRIMITIVES.....	29
8	SERVICE USER GUIDELINES ..	29
9	SERVICE PROVIDER-SPECIFIC INFORMATION.....	29
10	IDL.....	29
11	UML	42
11.1	TRANSCEIVER SETUP RELATIONSHIPS.....	42
11.2	HAVE QUICK RADIO MODE RELATIONSHIPS.....	43
11.3	HAVE QUICK MEDIA SETUP RELATIONSHIPS.....	44
11.4	HAVE QUICK RECEIVE COMMAND RELATIONSHIPS	45
11.5	NON-REAL-TIME SERVICE PROVIDER INHERITANCE.....	46
11.6	HAVE QUICK ULONGPACKET RELATIONSHIPS	47
11.7	HAVE QUICK USHORTPACKET RELATIONSHIPS	48
11.8	HAVE QUICK OCTETPACKET RELATIONSHIPS.....	49
11.9	REAL-TIME SERVICE PROVIDER INHERITANCE.....	50
11.10	REAL-TIME SERVICE USER INHERITANCE.....	50
11.11	HAVE QUICK COMPONENT DIAGRAM.....	51

List of Figures

Figure 1. JTRS Service Layers.....	1
Figure 2. TransceiverSetup	4
Figure 3. Have Quick Radio Mode	4
Figure 4. Have Quick RadioMode State Diagram.....	5
Figure 5. Have Quick MediaSetup	6
Figure 6. Have Quick ReceiveCommand.....	6
Figure 7. UlongPacket.....	7
Figure 8. UshortPacket.....	8
Figure 9. OctetPacket.....	9
Figure 10. HQ Packet Transfer (Tx/Rx).....	28
Figure 11. Transceiver Setup Relationships.....	42
Figure 12. Have Quick Radio Mode Relationships.....	43
Figure 13. Have Quick Media Setup Relationships	44
Figure 14. Have Quick Receive Command Relationships.....	45
Figure 15. Non-Real-Time Service Provider Inheritance	46
Figure 16. Have Quick UlongPacket Relationships.....	47
Figure 17. Have Quick UshortPacket Relationships.....	48
Figure 18. Have Quick OctetPacket Relationships	49
Figure 19. Real-Time Service Provider Inheritance.....	50
Figure 20. Real-Time Service User Inheritance.....	50
Figure 21. Have Quick Component Diagram.....	51

1 INTRODUCTION.

1.1 OVERVIEW.

The Have Quick Physical non-real time application-program interface (API) provides a standardized interface to the Physical Layer. The Physical Layer Services are grouped into real time and non-real time Service Groups. The non-real time physical Layer services provide Service Users with methods to send non-real-time configuration and control data into the Physical Layer. Real time control and real time signals are described in the Physical real time section of the Have Quick API.

Location of the Physical Layer with respect to other JTRS layers is shown in the following diagram. The Physical real-time interface is indicated by "A", while the non-real time interface is the "B" interface into the Physical Layer.

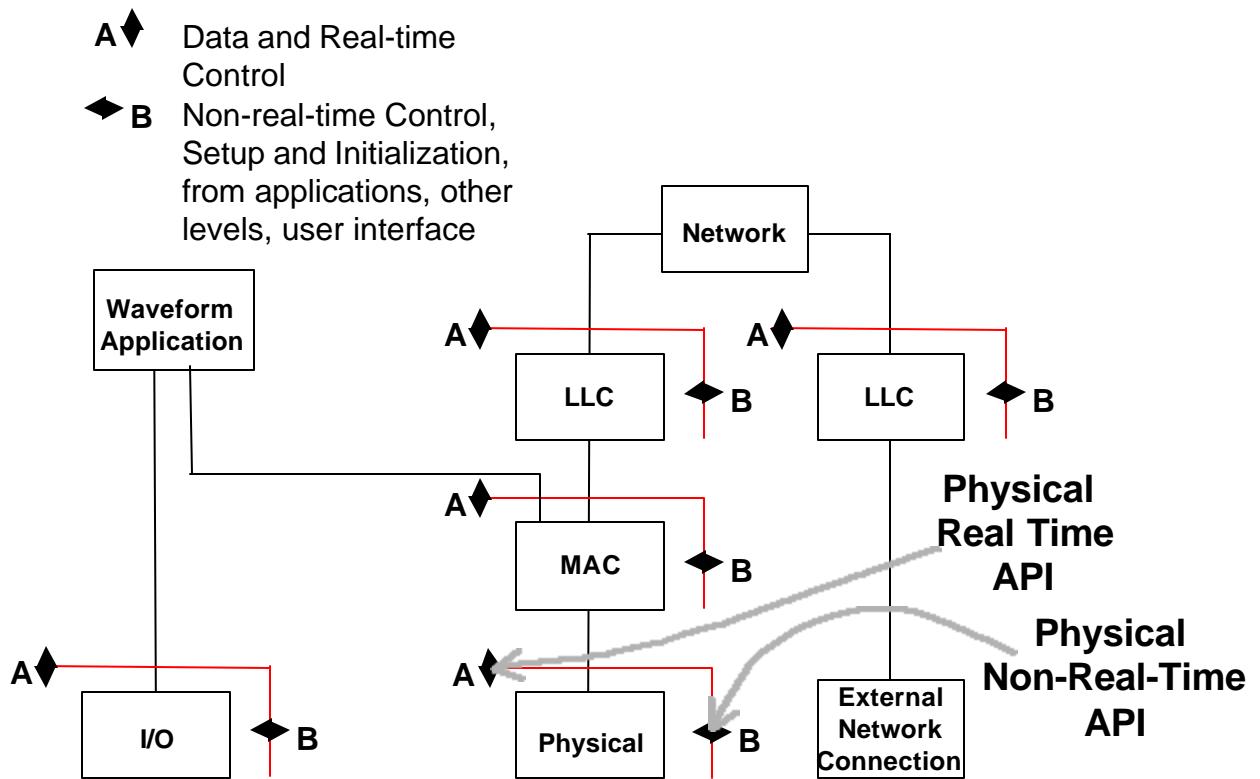


Figure 1. JTRS Service Layers

1.2 SERVICE LAYER DESCRIPTION.

The primary Physical Layer Service User is the MAC layer.

In general, the Physical Layer is responsible for configuration of the transmit/receive parameters, establishment of modulation type, control of transmit/receive, and the modulation and demodulation of data and for transmitting bits over-the-air.

1.2.1 Non-Real-Time Services.

The non-real-time part of the Have Quick API is realized by instantiating the Physical non-real-time Building Block for Have Quick. The following Service Groups are included in the Physical Layer for non-real-time.

- transceiver setup
- media setup
- radio mode
- receive command

Each Non-real-time Service Group is described in detail within Section 3.

1.2.2 Real-Time Services.

The real-time part of the Have Quick API is realized by instantiating the Physical real-time Building Block for Have Quick. The following Service Groups are included in the Physical Layer for real-time.

- transmit and receive packets and accompanying control

Each Real-time Service Group is described in detail within Section 3.

1.3 MODES OF SERVICE.

This API is used for all transmit and receive modes.

1.4 SERVICE STATES.

The API is used for all modes of operations except initialization.

1.5 REFERENCED DOCUMENTS.

Document No.

MSRC-5000SCA

Document Title

Software Communications Architecture
Specification

2 UUID.

The UUID for this API is 477d4dd0-d1d3-11d4-8cc8-00104b23b8a2.

3 SERVICES.

The Have Quick Physical interface is defined in terms of the services provided by the Service Provider, and the individual primitives that may flow between the Service User and Service Provider.

The services are tabulated below and described within the real-time and non-real-time service subsections.

SERVICE GROUP	SERVICE	PRIMITIVE
Transceiver Setup	Set Up Receiver Parameters	<i>setUpReceiverParams</i> (RecvParams: in TransceiverParams_Type) : boolean
	Set Up Transmitter Parameters	<i>setUpTransmitterParams</i> (TransParams: in TransceiverParams_Type) : boolean
	Set Date	<i> setDate(date: in HQ_API::HQ::DateType) : boolean</i>
Media Setup	Set Up Media Type	<i>setUpMediaType</i> (MediaParams : in MediaParams_Type) : boolean
Radio Mode	Set Radio Mode	<i>setRadioMode (RadioMode : in Mode_Type) : boolean</i>
Receive Command	Receive Command	<i>receive (control : RcvPacketControl) : void</i>
Transmit/Receive Packets	Packet BB Instantiated only. Not defined in this API. See Service Definition Description Generic Packet Service Building Block	<i>pushPacket(priority : in octet, control : in HQ_API::HQ::NULLControl, payload : in HQ_API::PortTypes::UlongSequence)</i>

3.1 NON-REAL-TIME SERVICES.

3.1.1 TransceiverSetup.

This class (Figure 2) sets up the parameters in the Physical Layer that are not modulation or real-time dependent.

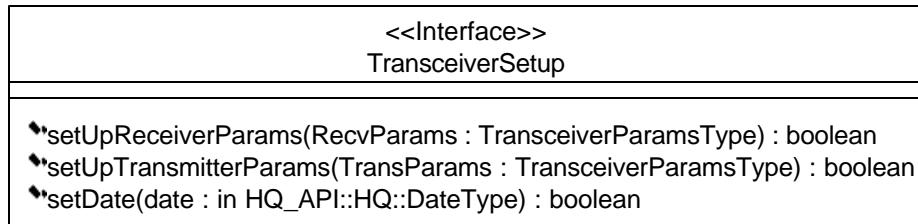


Figure 2. TransceiverSetup

3.1.1.1 setUpReceiveParams.

setUpReceiveParams sets the receive channel bandwidth to 25 kHz and sets the demodulator to AM.

3.1.1.2 setUpTransmit Params.

setUpTransmitParams sets the transmit channel bandwidth to 25 kHz and sets the modulator to AM.

3.1.1.3 setDate.

setDate sets the current day-of-year.

3.1.2 RadioMode

RadioMode (Figure 3) as it applies to the Physical Layer, sets the Physical Layer to off, standby, operational, or test mode.

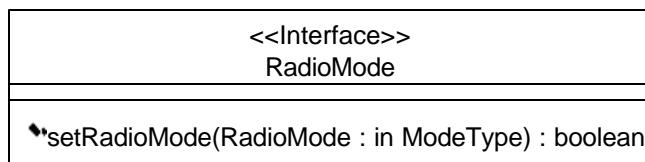


Figure 3. Have Quick Radio Mode

These modes are defined as follows:

Off

means that the waveform is no longer active on the transceiver. Power may be turned off to the Physical Layer, if possible. This is typically a low power mode.

Operate

is normal operation transmit and receive operations allowed.

Test

is a special mode of the radio to verify correct operation. Normal transmit and receive are disabled, but test transmit and receive can occur.

3.1.2.1 set RadioMode.

setRadioMode sets the Physical layer to either standby, operate, or off according the state diagram shown below.

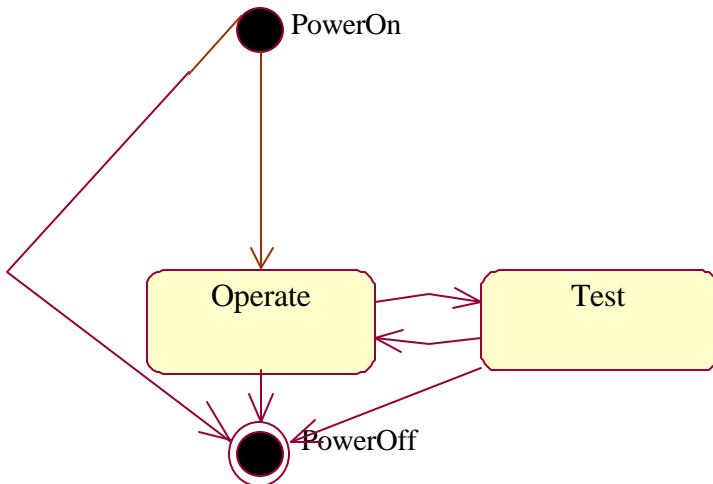


Figure 4. Have Quick RadioMode State Diagram

3.1.3 MediaSetup

MediaSetup (Figure 5) configures the squelch levels for Noise and AGC squelch settings.

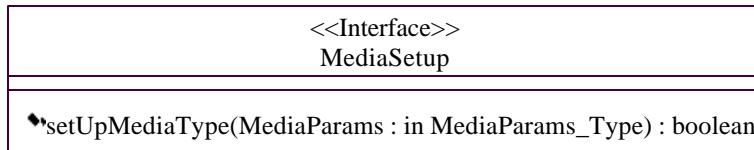


Figure 5. Have Quick MediaSetup

3.1.4 ReceiveCommand.

ReceiveCommand (Figure 6) initiates a receive at the physical device using the specified parameters.

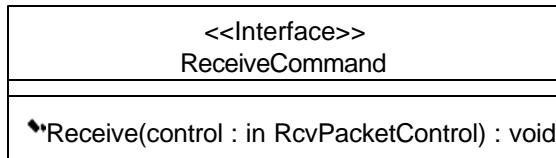


Figure 6. Have Quick ReceiveCommand

3.2 REAL-TIME SERVICES.

Real-time control and data will be pushed upstream and downstream using control and data packets. The data packets include a control header for transferring real time control information with the data. These services are obtained from the Packet Building Block. Refer to the SCA Service Definition for the Packet Building Block.

3.2.1 UlongPacket.

This interface (Figure 7) provides the attributes and operations required to support packet routing capabilities for the Have Quick waveform when data to be routed requires a data structure composed of unsigned long data elements.

<<Interface>>	
UlongPacket	
↳ maxPayloadSize : unsigned short	
↳ minPayloadSize : unsigned short	
• pushPacket(priority : in octet, control : in HQ_API::HQ::NullControl, payload : in HQ_API::PortTypes::UlongSequence) : void	
• spaceAvailable(priorityQueueID : in octet) : short	
• enableFlowControlSignals(enable : in boolean) : void	
• enableEmptySignal(enable : in boolean) : void	
• setNumOfPriorityQueues(numOfPriorities : in octet) : void	

Figure 7. UlongPacket

3.2.1.1 pushPacket.

This operation is used to push Client data to the Server with a Control element and a Payload element.

3.2.1.2 spaceAvailable.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queueing mechanism implemented for this waveform.

3.2.1.3 enableFlowControlSignals.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queueing mechanism implemented for this waveform.

3.2.1.4 enableEmptySignal.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queueing mechanism implemented for this waveform.

3.2.1.5 setNumOfPriorityQueues.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queueing mechanism implemented for this waveform.

3.2.2 UshortPacket.

This interface (Figure 8) provides the attributes and operations required to support packet routing capabilities for the Have Quick waveform when data to be routed requires a data structure composed of unsigned short data elements.

<<Interface>> UshortPacket
↳ maxPayloadSize : unsigned short ↳ minPayloadSize : unsigned short
• pushPacket(priority : in octet, control : in HQ_API::HQ::NullControl, payload : in HQ_API::PortTypes::UshortSequence) : void • spaceAvailable(priorityQueueID : in octet) : short • enableFlowControlSignals(enable : in boolean) : void • enableEmptySignal(enable : in boolean) : void • setNumOfPriorityQueues(numOfPriorities : in octet) : void

Figure 8. UshortPacket

3.2.2.1 pushPacket.

This operation is used to push Client data to the Server with a Control element and a Payload element.

3.2.2.2 spaceAvailable.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queueing mechanism implemented for this waveform.

3.2.2.3 enableFlowControlSignals.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queueing mechanism implemented for this waveform.

3.2.2.4 enableEmptySignal.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queueing mechanism implemented for this waveform.

3.2.2.5 setNumOfPriorityQueues.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queueing mechanism implemented for this waveform.

3.2.3 OctetPacket.

This interface (Figure 9) provides the attributes and operations required to support packet routing capabilities for the Have Quick waveform when data to be routed requires an data structure composed of a stream of octets.

<<Interface>>
OctetPacket
↳ maxPayloadSize : unsigned short
↳ minPayloadSize : unsigned short
• pushPacket(priority : in octet, control : in HQ_API::HQ::NullControl, payload : in HQ_API::CF::OctetSequence) : void
• spaceAvailable(priorityQueueID : in octet) : short
• enableFlowControlSignals(enable : in boolean) : void
• enableEmptySignal(enable : in boolean) : void
• setNumOfPriorityQueues(numOfPriorities : in octet) : void

Figure 9. OctetPacket

3.2.3.1 pushPacket.

This operation is used to push Client data to the Server with a Control element and a Payload element.

3.2.3.2 spaceAvailable.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queueing mechanism implemented for this waveform.

3.2.3.3 enableFlowControlSignals.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queueing mechanism implemented for this waveform.

3.2.3.4 enableEmptySignal.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queueing mechanism implemented for this waveform.

3.2.3.5 setNumOfPriorityQueues.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queueing mechanism implemented for this waveform.

4 SERVICE PRIMITIVES.

4.1 TRANSCEIVER SETUP.

4.1.1 setUpReceiverParams

This primitive sets up the parameters in the Physical Layer of the transceiver that are specific to receive.

4.1.1.1 Synopsis.

```
boolean setUpReceiverParams(
    in TransceiverParamsType          RecvParams
)
)
```

4.1.1.2 Parameters.

RecvParams

```
struct TransceiverParamsType{
    unsigned short      transceiverID;
    CapabilityType     capability;
    unsigned long       frequency;
}
```

transceiverID

is the identifier of the current transceiver device.

capability

indicates the capability parameters associated with the command.

frequency

indicates the current transmit frequency.

capability

```
struct CapabilityType{
    boolean            receive;
    boolean            transmit;
    boolean           sendModeMsg;
    boolean           CVSD;
}
```

receive

the boolean that indicates if these variables are to be set for the receiver device.(TRUE in this case)

transmit

is the boolean that indicates if these variables are to be set for the transmitter device. (FALSE in this case).

sendModeMsg

is the boolean that indicates if the Status messages are to be routed to the host from the modem device.

CVSD

is the boolean that indicates if CVSD is to be employed.

4.1.1.3 State.

This command valid in all states.

4.1.1.4 New State.

This command causes a state change.

4.1.1.5 Response.

TRUE if Receiver Parameters is set as specified in the call, FALSE otherwise.

4.1.1.6 Originator.

The service user. Typically these parameters are set through the Config/Querry interface.

4.1.1.7 Errors/Exceptions.

TBS

4.1.2 setUpTransmitterParams.

This primitive sets up the parameters in the Physical Layer of the transceiver that are specific to transmit.

4.1.2.1 Synopsis.

```
boolean setUpTransmitterParams(
    in TransceiverParamsType           TransParams
)
```

4.1.2.2 Parameters.

TransParams

```
struct TransceiverParamsType{
    unsigned short      transceiverID;
    CapabilityType     capability;
    unsigned long       frequency;
```

}

transceiverID

is the identifier of the current transceiver device.

capability

indicates the capability parameters associated with the command.

frequency

indicates the current transmit frequency.

capability

```
struct CapabilityType{  
    boolean receive;  
    boolean transmit;  
    boolean sendModeMsg;  
    boolean CVSD;  
}
```

receive

the boolean that indicates if these variables are to be set for the receiver device (FALSE in this case).

transmit

is the boolean that indicates if these variables are to be set for the transmitter device (TRUE in this case).

sendModeMsg

is the boolean that indicates if the Status messages are to be routed to the host from the modem device.

CVSD

is the boolean that indicates if CVSD is to be employed.

4.1.2.3 State.

This command valid in all states.

4.1.2.4 New State.

This command does cause a state change.

4.1.2.5 Response.

TRUE if the Transmitter Parameters are set as specified in the call; FALSE otherwise.

4.1.2.6 Originator.

The service user. Typically, these parameters are set through the Config/Querry interface.

4.1.2.7 Errors/Exceptions.

TBS

4.1.3 setDate.

This primitive sets the current day-of-year and year attributes.

4.1.3.1 Synopsis.

```
boolean setDate (
    in HQ_API::HQ::DateType      date
)
```

4.1.3.2 Parameters.

date

```
struct DateType{
    long          dayOfYear;
    long          year;
}
```

dayOfYear

is the current integer day of year.

year

is the current integer year.

4.1.3.3 State.

This command valid in all states.

4.1.3.4 New State.

This command does cause a state change.

4.1.3.5 Response.

TRUE if the Transmitter Parameters are set as specified in the call; FALSE otherwise.

4.1.3.6 Originator.

The service user. Typically, these parameters are set through the Config/Querry interface.

4.1.3.7 Errors/Exceptions.

TBS

4.2 RADIOMODE.

4.2.1 setRadioMode

This primitive sets the mode of the radio Physical Layer to Off, Operate, or Test.

4.2.1.1 Synopsis.

```
boolean setRadioMode(  
    in ModeType          RadioMode ;  
)
```

4.2.1.2 Parameters.

RadioMode

sets the radio to be one of four enumerated modes.

```
enum ModeType{  
    Off;  
    Operate;  
    Test;  
}
```

Off

means that the waveform is no longer active on the transceiver.

Operate

is normal operation transmit and receive operations allowed.

Test

is a special mode of the radio to verify correct operation. Normal transmit and receive are disabled, but test transmit and receive can occur.

4.2.1.3 State.

This command operates in all states.

4.2.1.4 New State.

The methods cause changes in state. The new state is specified in the command.

4.2.1.5 Response.

TRUE if the Radio Mode is set correctly; FALSE otherwise.

4.2.1.6 Originator.

This primitive is initiated by the Service User, normally through the Human Computer Interface(HCI).

4.2.1.7 Errors/Exceptions.

None.

4.3 MEDIASETUP.

This interface provides the attributes and operations required to support squelch control capabilities for the Have Quick waveform.

4.3.1.1 Synopsis.

```
boolean setUpMediaType(  
    in MediaParams_Type      MediaParams ;  
)
```

4.3.1.2 Parameters.

MediaParams

```
struct MediaParams_Type{  
    unsigned long      squelchLevelOffToOn;  
    SquelchType       squelchType;  
}
```

```
enum SquelchType{  
    AGC;  
    Noise;  
}
```

squelchLevelOffToOn

indicates the desired level at which to apply squelch.

squelchType

indicates whether the squelch is of the AGC or Noise type.

4.3.1.3 State.

This command operates in all states.

4.3.1.4 New State.

The methods cause changes in state. The new state is specified in the command.

4.3.1.5 Response.

TRUE if the Radio Mode is set correctly; FALSE otherwise.

4.3.1.6 Originator.

This primitive is initiated by the Service User, normally through the Human Computer Interface(HCI).

4.3.1.7 Errors/Exceptions.

None.

4.4 ULONGPACKET.

This interface provides the attributes and operations required to support packet routing capabilities for the Have Quick waveform when data to be routed requires a data structure composed of unsigned long data elements.

4.4.1 pushPacket.

This operation is used to push Client data to the Server with a Control element and a Payload element.

4.4.1.1 Synopsis.

```
void pushPacket(priority : in octet, control : in HQ_API::HQ::NullControl, payload : in HQ_API::PortTypes::UlongSequence)
```

4.4.1.2 Parameters.

Priority

Priority is a single octet. This parameter is not used for a HQ implementation. It shall be set to 0 to indicate a Null priority.

Control

Control is of type NullControl. This parameter is not used for a HQ implementation.

Payload

Payload is of type UlongSequence. This parameter represents the sequence of the data stream to be pushed as a UlongSequence.

4.4.1.3 State.

TBD

4.4.1.4 New State.

TBD

4.4.1.5 Response.

TBD

4.4.1.6 Originator.

Service User.

4.4.1.7 Errors/Exceptions.

None.

4.4.2 spaceAvailable.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queueing mechanism implemented for this waveform.

4.4.2.1 Synopsis.

short spaceAvailable (priorityQueueId : in octet)

4.4.2.2 Parameters.

PriorityQueueId

PriorityQueueId is of type octect. This parameter is not used for a HQ implementation.

4.4.2.3 State.

TBD

4.4.2.4 New State.

TBD

4.4.2.5 Response.

TBD

4.4.2.6 Originator.

Service User

4.4.2.7 Errors/Exceptions.

None.

4.4.3 enableFlowControlSignals.

This operation shall be overridden and shall specify no functionality for HQ operations.
There is no queueing mechanism implemented for this waveform.

4.4.3.1 Synopsis.

void enableFlowControlSignals (enable : in boolean)

4.4.3.2 Parameters.

enable

Enable is of type boolean. This parameter is not used for a HQ implementation.

4.4.3.3 State.

TBD

4.4.3.4 New State.

TBD

4.4.3.5 Response.

TBD

4.4.3.6 Originator.

Service User

4.4.3.7 Errors/Exceptions.

None.

4.4.4 enableEmptySignal.

This operation shall be overridden and shall specify no functionality for HQ operations.
There is no queueing mechanism implemented for this waveform.

4.4.4.1 Synopsis.

void enableEmptyControlSignal (enable : in boolean)

4.4.4.2 Parameters.

enable

Enable is of type boolean. This parameter is not used for a HQ implementation.

4.4.4.3 State.

TBD

4.4.4.4 New State.

TBD

4.4.4.5 Response.

TBD

4.4.4.6 Originator.

Service User.

4.4.4.7 Errors/Exceptions.

None.

4.4.5 setNumberOfPriorityQueues.

This operation shall be overridden and shall specify no functionality for HQ operations.
There is no queueing mechanism implemented for this waveform.

4.4.5.1 Synopsis.

void setNumberOfPriorityQueues (numOfPriorities : in octet)

4.4.5.2 Parameters.

numOfPriorities

numOfPriorities is of type octet. This parameter is not used for a HQ implementation.

4.4.5.3 State.

TBD

4.4.5.4 New State.

TBD

4.4.5.5 Response.

TBD

4.4.5.6 Originator.

Service User.

4.4.5.7 Errors/Exceptions.

None.

4.5 USHORTPACKET.

This interface provides the attributes and operations required to support packet routing capabilities for the Have Quick waveform when data to be routed requires a data structure composed of unsigned short data elements.

4.5.1 pushPacket.

This operation is used to push Client data to the Server with a Control element and a Payload element.

4.5.1.1 Synopsis.

```
void pushPacket(priority : in octet, control : in HQ_API::HQ::NullControl, payload : in HQ_API::PortTypes::UshortSequence)
```

4.5.1.2 Parameters.

Priority

Priority is a single octet. This parameter is not used for a HQ implementation. It shall be set to 0 to indicate a Null priority.

Control

Control is of type NullControl. This parameter is not used for a HQ implementation.

Payload

Payload is of type uShortSequence. This parameter represents the sequence of the data stream to be pushed as a UshortSequence.

4.5.1.3 State.

TBD

4.5.1.4 New State.

TBD

4.5.1.5 Response.

TBD

4.5.1.6 Originator.

Service User.

4.5.1.7 Errors/Exceptions.

None.

4.5.2 spaceAvailable.

This operation shall be overridden and shall specify no functionality for HQ operations.
There is no queueing mechanism implemented for this waveform.

4.5.2.1 Synopsis.

short spaceAvailable (priorityQueueId : in octet)

4.5.2.2 Parameters.

PriorityQueueId

PriorityQueueId is of type octet. This parameter is not used for a HQ implementation.

4.5.2.3 State.

TBD

4.5.2.4 New State.

TBD

4.5.2.5 Response.

TBD

4.5.2.6 Originator.

Service User.

4.5.2.7 Errors/Exceptions.

None.

4.5.3 enableFlowControlSignals.

This operation shall be overridden and shall specify no functionality for HQ operations.
There is no queueing mechanism implemented for this waveform.

4.5.3.1 Synopsis.

void enableFlowControlSignals (enable : in boolean)

4.5.3.2 Parameters.

enable

Enable is of type boolean. This parameter is not used for a HQ implementation.

4.5.3.3 State.

TBD

4.5.3.4 New State.

TBD

4.5.3.5 Response.

TBD

4.5.3.6 Originator.

Service User.

4.5.3.7 Errors/Exceptions.

None.

4.5.4 enableEmptySignal.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queueing mechanism implemented for this waveform.

4.5.4.1 Synopsis.

void enableEmptySignal (enable : in boolean)

4.5.4.2 Parameters.

enable

Enable is of type boolean. This parameter is not used for a HQ implementation.

4.5.4.3 State.

TBD

4.5.4.4 New State.

TBD

4.5.4.5 Response.

TBD

4.5.4.6 Originator.

Service User.

4.5.4.7 Errors/Exceptions.

None.

4.5.5 setNumOfPriorityQueues.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queueing mechanism implemented for this waveform.

4.5.5.1 Synopsis.

void setNumberOfPriorityQueues (numOfPriorities : in octet)

4.5.5.2 Parameters.

enable

Enable is of type boolean. This parameter is not used for a HQ implementation.

4.5.5.3 State.

TBD

4.5.5.4 New State.

TBD

4.5.5.5 Response.

TBD

4.5.5.6 Originator.

Service User.

4.5.5.7 Errors/Exceptions.

None.

4.6 OCTETPACKET.

This interface provides the attributes and operations required to support packet routing capabilities for the Have Quick waveform when data to be routed requires an data structure composed of a stream of octets.

4.6.1 pushPacket.

This operation is used to push Client data to the Server with a Control element and a Payload element.

4.6.1.1 Synopsis.

void pushPacket(priority : in octet, control : in HQ_API::HQ::NullControl, payload : in HQ_API::PortTypes::OctetSequence)

4.6.1.2 Parameters.

Priority

Priority is a single octet. This parameter is not used for a HQ implementation. It shall be set to 0 to indicate a Null priority.

Control

Control is of type NullControl. This parameter is not used for a HQ implementation.

Payload

Payload is of type OctetSequence. This parameter represents the sequence of the data stream to be pushed as a OctetSequence.

4.6.1.3 State.

TBD

4.6.1.4 New State.

TBD

4.6.1.5 Response.

TBD

4.6.1.6 Originator.

Service User.

4.6.1.7 Errors/Exceptions.

None.

4.6.2 spaceAvailable.

This operation shall be overridden and shall specify no functionality for HQ operations. There is no queueing mechanism implemented for this waveform.

4.6.2.1 Synopsis.

short spaceAvailable (priorityQueueID : in octet)

4.6.2.2 Parameters.

PriorityQueueID

PriorityQueueId is of type octet. This parameter is not used for a HQ implementation.

4.6.2.3 State.

TBD

4.6.2.4 New State.

TBD

4.6.2.5 Response.

TBD

4.6.2.6 Originator.

Service User.

4.6.2.7 Errors/Exceptions.

None.

4.6.3 enableFlowControlSignals.

This operation shall be overridden and shall specify no functionality for HQ operations.
There is no queueing mechanism implemented for this waveform.

4.6.3.1 Synopsis.

void enableFlowControlSignals (enable : in boolean)

4.6.3.2 Parameters.

enable

Enable is of type boolean. This parameter is not used for a HQ implementation.

4.6.3.3 State.

TBD

4.6.3.4 New State.

TBD

4.6.3.5 Response.

TBD

4.6.3.6 Originator.

Service User.

4.6.3.7 Errors/Exceptions.

None.

4.6.4 enableEmptySignal.

This operation shall be overridden and shall specify no functionality for HQ operations.
There is no queueing mechanism implemented for this waveform.

4.6.4.1 Synopsis.

void enableEmptySignal (enable : in boolean)

4.6.4.2 Parameters.

enable

Enable is of type boolean. This parameter is not used for a HQ implementation.

4.6.4.3 State.

TBD

4.6.4.4 New State.

TBD

4.6.4.5 Response.

TBD

4.6.4.6 Originator.

Service User.

4.6.4.7 Errors/Exceptions.

None.

4.6.5 setNumberOfPriorityQueues.

This operation shall be overridden and shall specify no functionality for HQ operations.
There is no queueing mechanism implemented for this waveform.

4.6.5.1 Synopsis.

void setNumberOfPriorityQueues (numberOfPriorities :in octet)

4.6.5.2 Parameters.

enable

Enable is of type boolean. This parameter is not used for a HQ implementation.

4.6.5.3 State.

TBD

4.6.5.4 New State.

TBD

4.6.5.5 Response.

TBD

4.6.5.6 Originator.

Service User.

4.6.5.7 Errors/Exceptions.

None.

4.7 RECEIVECOMMAND.

This interface provides the operation required to support configuration/control of HQ packet reception according to the defined RcvPacketControl attributes.

4.7.1 Receive.

This operation is used to command the Servant to receive a HQ packet according to the current RcvPacketControl attributes.

4.7.1.1 Synopsis.

void Receive (control: in RcvPacketControl)

4.7.1.2 Parameters.

control

```
struct RcvPacketControl{  
    unsigned long frequencyInHz;  
    ModulationType modulation;  
}
```

frequencyInHz

is the receive frequency to use.

modulation

is AM for Have Quick.

4.7.1.3 State.

TBD

4.7.1.4 New State.

TBD

4.7.1.5 Response.

TBD

4.7.1.6 Originator.

Service User.

4.7.1.7 Errors/Exceptions.

None.

5 ALLOWABLE SEQUENCE OF SERVICE PRIMITIVES.

There are no defined sequences of primitives. The following figure depicts a typical use case scenario.

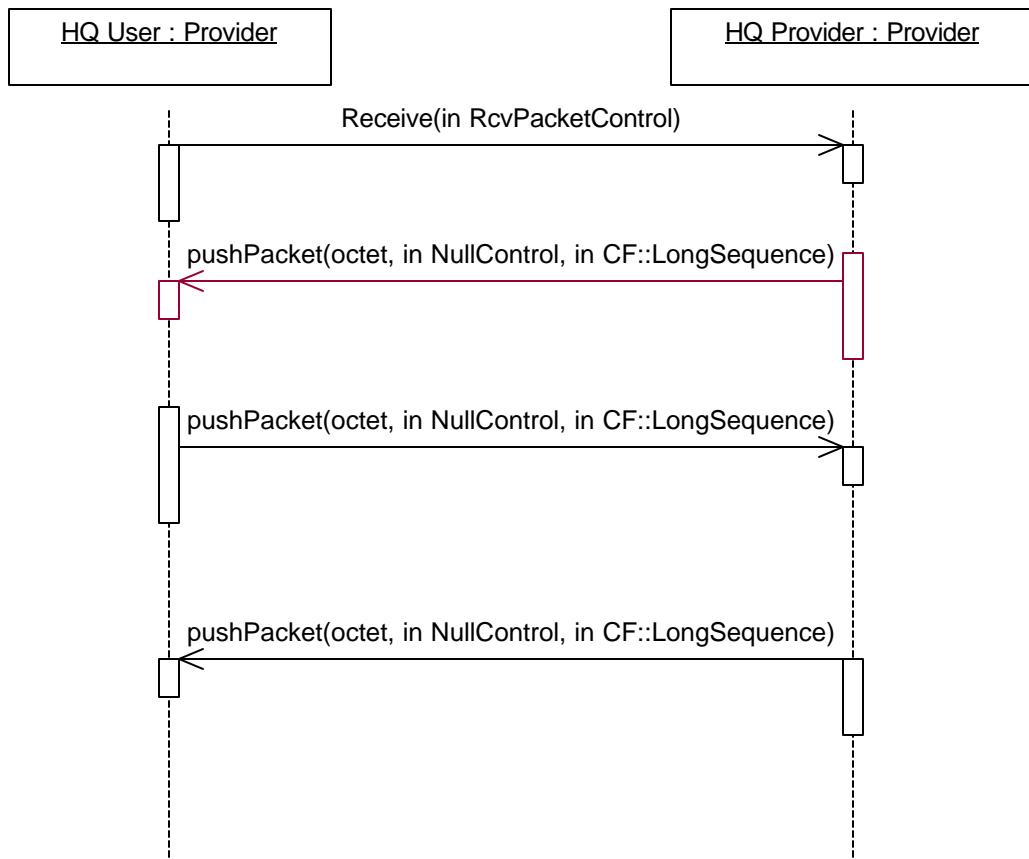


Figure 10. HQ Packet Transfer (Tx/Rx)

6 UTILIZATION OF PHYSICAL NON-REAL-TIME BUILDING BLOCKS.

Refer to the following table for services and operations.

Services	Operations
Transceiver Setup	setReceiverParmeters setTransmterParameters
Media Setup	setUpMediaType
RadioMode	setRadioMode

7 PRECEDENCE OF SERVICE PRIMITIVES.

There is no precedence of primitives.

8 SERVICE USER GUIDELINES.

There are no Service User Guidelines for the Have Quick Physical API provided.

9 SERVICE PROVIDER-SPECIFIC INFORMATION.

There is no Service Provider-Specific Information for the Have Quick Physical API.

10 IDL.

The Have Quick interface design depicted in IDL source code is shown below.

```
//Source file: c:/program files/devstudio/vc/atl/include/HQ.idl

#ifndef __HQ_DEFINED
#define __HQ_DEFINED

/* CmIdentification
 %X% %Q% %Z% %W% */

module HQ {

    /* Control of packet for los with nothing in it. */
    struct NullControl {
    };

    struct DateType {
        long dayOfYear;
    };
}
```

```

        long year;
    };

    module Physical {

        enum squelchType {
            AGC,
            NOISE
        };

        struct MediaParams_Type {
            squelchType squelchType;
            /* Gen_Set_Squelch_Msg #20
             * Gen_AGC_Squelch_Msg #87 */
            unsigned long squelchLevelOffToOn;
        };

        interface MediaSetup {
            /*
             * @roseuid 39F85AA6016C */
            boolean setUpMediaType (
                in MediaParams_Type MediaParams
            );
        };

        /* Provide a pushPacket with a octet payload */

        interface OctetPacket {
            /* The maxPacketSize is a read only attribute set by
             * the Packet Server and the get operation reports back
             * the maximum number of traffic units allowed in one
             * pushPacket call. */

            attribute unsigned short maxPayloadSize;
            attribute unsigned short minPayloadSize;

            /* This operation is used to push Client data to the
             * Server with a Control element and a Payload element.
             * @roseuid 39F8739C037A */
            void pushPacket (
                in octet priority,
                in HQ::NullControl control,
                in CF::OctetSequence payload
            );

            /* The operation returns the space available in the
             * Servers queue(s) in terms of the implementers defined
             * Traffic Units.
             * @roseuid 39F8739C037E */
            short spaceAvailable (
                in octet priorityQueueID
            );

            /* This operation allows the client to turn the High
             * Watermark Signal ON and OFF.
        };
    };
}
```

```

@roseuid 39F8739C0380 */
void enableFlowControlSignals (
    in boolean enable
);

/* This operation allows the client to turn theEmpty
Signal ON and OFF.
@roseuid 39F8739C0382 */
void enableEmptySignal (
    in boolean enable
);

/*
@roseuid 39F8739C0384 */
void setNumOfPriorityQueues (
    in octet numOfPriorities
);

};

/* Packet */

interface UlongPacket {
    /* The maxPacketSize is a read only attribute set by
    the Packet Server and the get operation reports back
    the maximum number of traffic units allowed in one
    pushPacket call. */

    attribute unsigned short maxPayloadSize;
    attribute unsigned short minPayloadSize;

    /* This operation is used to push Client data to the
    Server with a Control element and a Payload element.
    @roseuid 39F875FC01E0 */
    void pushPacket (
        in octet priority,
        in HQ::NullControl control,
        in PortTypes::UlongSequence payload
    );

    /* The operation returns the space available in the
    Servers queue(s) in terms of the implementers defined
    Traffic Units.
    @roseuid 39F875FC01FE */
    short spaceAvailable (
        in octet priorityQueueID
    );

    /* This operation allows the client to turn the High
    Watermark Signal ON and OFF.
    @roseuid 39F875FC0208 */
    void enableFlowControlSignals (
        in boolean enable
    );
}

```

```

/* This operation allows the client to turn theEmpty
Signal ON and OFF.
@roseuid 39F875FC021C */
void enableEmptySignal (
    in boolean enable
);

/*
@roseuid 39F875FC0227 */
void setNumOfPriorityQueues (
    in octet numOfPriorities
);

};

/* Packet */

interface UshortPacket {
    /* The maxPacketSize is a read only attribute set by
       the Packet Server and the get operation reports back
       the maximum number of traffic units allowed in one
       pushPacket call. */

    attribute unsigned short maxPayloadSize;
    attribute unsigned short minPayloadSize;

    /* This operation is used to push Client data to the
       Server with a Control element and a Payload element.
    @roseuid 39F87994028D */
    void pushPacket (
        in octet priority,
        in HQ::NullControl control,
        in PortTypes::UshortSequence payload
    );

    /* The operation returns the space available in the
       Servers queue(s) in terms of the implementers defined
       Traffic Units.
    @roseuid 39F8799402AC */
    short spaceAvailable (
        in octet priorityQueueID
    );

    /* This operation allows the client to turn the High
       Watermark Signal ON and OFF.
    @roseuid 39F8799402B5 */
    void enableFlowControlSignals (
        in boolean enable
    );

    /* This operation allows the client to turn theEmpty
       Signal ON and OFF.
    @roseuid 39F8799402B7 */
    void enableEmptySignal (
        in boolean enable
    );
}

```

```

/*
@roseuid 39F8799402C0 */
void setNumOfPriorityQueues (
    in octet numOfPriorities
);

};

enum ModeType {
    Off,                      /* HQ_Stop_Msg #6 */
    Operate,                  /* HQ_Start_Msg #3 */
    Test                      /* DSP_Run_Bit_Msg #34 */
};

/* Radio Mode */

interface RadioMode {
    /*
    @roseuid 39F87E38030F */
    boolean setRadioMode (
        in ModeType RadioMode
    );
};

interface User : UlongPacket, PacketBB::PacketSignals {
};

/* This type is a CORBA unbounded sequence of octets. */

typedef sequence <octet> OctetSequence;

/* LOS modulation type */

enum ModulationType {
    AM
};

struct RcvPacketControl {
    /* The frequency to tune for the duration of the
    hop.. */
    unsigned long long frequencyInHz;
    /* Enumerated value that specifies the type of
    demodulation or modulation to use . */
    ModulationType modulation;
};

/* Receive Command */

interface ReceiveCommand {
    /*
    @roseuid 39F8815E01E1 */
    void Receive (
        in RcvPacketControl control
    );
};

```

```

};

interface Provider : ReceiveCommand, UlongPacket,
    PacketBB::PacketSignals {
};

/* DSP_Capabilities_Msg #33 */

struct CapabilityType {
    boolean receive;
    boolean transmit;
    boolean sendModeMsg;
    boolean CVSD;
};

struct TransceiverParamsType {
    unsigned short transceiverID;
    /* Gen_Set_Frequency_Msg #23 */
    unsigned long frequency;
    CapabilityType capability;
};

/* Transciever Setup */

interface TransceiverSetup {
    /*
     * @roseuid 39F87F0501A1 */
    boolean setUpReceiverParams (
        TransceiverParamsType RcvParams
    );

    /*
     * @roseuid 39F87F0501BE */
    boolean setUpTransmitterParams (
        TransceiverParamsType TransParams
    );

    /*
     * @roseuid 3A09AFE30395 */
    boolean setDate (
        in HQ::DateType date
    );
};

interface Controller : TransceiverSetup, MediaSetup,
RadioMode, CF::Resource {
};

};

module MAC {

    /* Mac octet  Packet */
}

```

```

interface OctetPacket {
    /* The maxPacketSize is a read only attribute set by
       the Packet Server and the get operation reports back
       the maximum number of traffic units allowed in one
       pushPacket call. */

    attribute unsigned short maxPayloadSize;
    attribute unsigned short minPayloadSize;

    /* This operation is used to push Client data to the
       Server with a Control element and a Payload element.
       @roseuid 39F998E602BA */
    void pushPacket (
        in octet priority,
        in HQ::NullControl control,
        in CF::OctetSequence payload
    );

    /* The operation returns the space available in the
       Servers queue(s) in terms of the implementers defined
       Traffic Units.
       @roseuid 39F998E602E3 */
    short spaceAvailable (
        in octet priorityQueueID
    );

    /* This operation allows the client to turn the High
       Watermark Signal ON and OFF.
       @roseuid 39F998E602EC */
    void enableFlowControlSignals (
        in boolean enable
    );

    /* This operation allows the client to turn theEmpty
       Signal ON and OFF.
       @roseuid 39F998E602EE */
    void enableEmptySignal (
        in boolean enable
    );

    /*
       @roseuid 39F998E602F7 */
    void setNumOfPriorityQueues (
        in octet numOfPriorities
    );
};

/* MAC unsigned long packet */

interface UlongPacket {
    /* The maxPacketSize is a read only attribute set by
       the Packet Server and the get operation reports back
       the maximum number of traffic units allowed in one
       pushPacket call. */
}

```

```

attribute unsigned short maxPayloadSize;
attribute unsigned short minPayloadSize;

/* This operation is used to push Client data to the
Server with a Control element and a Payload element.
@roseuid 39F99B57021A */
void pushPacket (
    in octet priority,
    in HQ::NullControl control,
    in PortTypes::UlongSequence payload
);

/* The operation returns the space available in the
Servers queue(s) in terms of the implementers defined
Traffic Units.
@roseuid 39F99B57022D */
short spaceAvailable (
    in octet priorityQueueID
);

/* This operation allows the client to turn the High
Watermark Signal ON and OFF.
@roseuid 39F99B570237 */
void enableFlowControlSignals (
    in boolean enable
);

/* This operation allows the client to turn theEmpty
Signal ON and OFF.
@roseuid 39F99B570241 */
void enableEmptySignal (
    in boolean enable
);

/*
@roseuid 39F99B570243 */
void setNumOfPriorityQueues (
    in octet numOfPriorities
);

};

/* MAC unsigned short Packet */

interface UshortPacket {
    attribute unsigned short minPayloadSize;
    /* The maxPacketSize is a read only attribute set by
    the Packet Server and the get operation reports back
    the maximum number of traffic units allowed in one
    pushPacket call. */
    attribute unsigned short maxPayloadSize;

    /* This operation is used to push Client data to the
    Server with a Control element and a Payload element.
    @roseuid 39F99C500357 */
}

```

```

void pushPacket (
    in octet priority,
    HQ::NullControl control,
    in PortTypes::UshortSequence payload
);

/* The operation returns the space available in the
Servers queue(s) in terms of the implementers defined
Traffic Units.
@roseuid 39F99C500363 */
short spaceAvailable (
    in octet priorityQueueID
);

/* This operation allows the client to turn the High
Watermark Signal ON and OFF.
@roseuid 39F99C50036C */
void enableFlowControlSignals (
    in boolean enable
);

/* This operation allows the client to turn theEmpty
Signal ON and OFF.
@roseuid 39F99C500375 */
void enableEmptySignal (
    in boolean enable
);

/*
@roseuid 39F99C500377 */
void setNumOfPriorityQueues (
    in octet numOfPriorities
);

};

interface User : UlongPacket, PacketBB::PacketSignals {
};

interface Provider : UlongPacket, PacketBB::PacketSignals {
};

enum ErrorControlType {
    Active,
    Inactive
};

/* MAC Channel Error Control */

interface ChannelErrorControl {
    /*
    @roseuid 39F9A91002BA */
    void setChannelErrorControl (
        in ErrorControlType ErrorControl
    );
}

```

```

};

struct PowerModeType {
    /* Gen_Set_Power_Msg #24 */
    short powerLevel;
};

enum DataRateType {
    D75,
    D150,
    D300,
    D600,
    D1200,
    D2400,
    D4800,
    D16000,
    D1200N,
    D2400N,
    D4800N,
    D9600N,
    DOFF,
    RS232-9600,
    RS232-14400,
    RS232-38800,
    PKT,
    TF
};

enum ModeConfigType {
    NORMAL,                      /* Single Channel Mode
                                    HQ_Normal_Msg #8 */
    ACTIVE,                       /* Frequency Hop Mode
                                    HQ_Active_Msg #7 */
};

enum OperationModeType {
    LNE_Off,
    LNE_On,
    None
};

enum COMSECType {
    PT,                          /* Gen_Plain_Text_Mode_Msg #32 */
    CT                           /* Gen_Cipher_Text_Mode_Msg #31 */
};

struct MWODFillType {
    PortTypes::UlongSequence MWODData;
};

struct TODSeedType {
    HQ::DateType date;
    short hours;
    short minutes;
    short seconds;
    boolean immediate;
};

```

```

};

struct NetNumberType {
    short netNumber;
    short netType;
};

struct FMTFillType {
    PortTypes::UlongSequence FMTFrequencyData;
};

/* MAC TRANSEC */

interface TRANSEC {
    attribute short maxFILLLength;
    /* Not used */

    attribute short maxSEEDLength;
    attribute boolean zeroized;
    attribute short FMTFreqSize;

    /*
     * @roseuid 39F9C0E3026A */
    boolean loadFill (
        in short PresetNum,
        in MWODFillType MWODFill
    );

    /*
     * @roseuid 39F9C0E3027E */
    boolean loadSeed (
        in short seedNum,
        in TODSeedType TODSeed
    );

    /*
     * @roseuid 39F9C0E3030B */
    boolean readSeed (
        in short seedNum,
        out TODSeedType TODSeed
    );

    /*
     * @roseuid 39F9C0E30315 */
    boolean zeroize ();

    /*
     * @roseuid 3A0846C20200 */
    boolean loadNetNumber (
        in short PresetNum,
        in NetNumberType netData
    );

    /*
     * @roseuid 39F9C0E30274 */
    boolean readNetNumber (

```

```

        in short PresetNum,
        out NetNumberType netData
    );

/*
@roseuid 3A0966D501F8 */
boolean loadFMTFrequencies (
    in FMTFillType FMTFill
);

/*
@roseuid 3A0967020333 */
boolean readFMTFrequencies (
    out FMTFillType FMTFill
);

};

enum DataModeConfigType {
    Audio,           /* Gen_Audio_Mode_Msg #27 */
    Data             /* Gen_Data_Mode_Msg #28 */
};

struct ChannelModeType {
    ModeConfigType ModeConfig;
    DataRateType DataRate;
    COMSECType COMSEC;
    OperationModeType OperationMode;
    DataModeConfigType DataModeConfig;
    short ModemChannelNumber;
    boolean transmitInhibit;
};

/* Mac common utility */

interface MACCommonUtility {
    attribute unsigned long minTU;
    attribute unsigned long maxTU;

    /*
@roseuid 39F9AA430303 */
    boolean activateChannel (
        in short PresetNum
    );

    /*
@roseuid 39F9AA430316 */
    boolean setRFPowerControl (
        in PowerModeType PowerLevel
    );

    /*
@roseuid 39F9AA430321 */
    boolean setMode (
        in ChannelModeType ChannelMode
    );
}

```

```
/*
@roseuid 39F9AA43032A */
void getMinTU (
    out unsigned long MinTU
);

/*
@roseuid 39F9AA43032C */
void getMaxTU (
    out unsigned long MaxTU
);

/*
@roseuid 3A0975A70009 */
boolean transmitTOD ();

/*
@roseuid 3A0975A7006D */
boolean receiveTOD ();

/*
@roseuid 3A0975A700C7 */
boolean stopRectOD ();

/*
@roseuid 3A09ACA80237 */
boolean setXmitMode (
    in boolean PTTIndicator //Gen_PTT_Msg #76
);

};

interface Controller : CF::Resource, ChannelErrorControl,
    MACCommonUtility, TRANSEC {
};

};

#endif
```

11 UML.

UML class and component diagrams are shown in Figures 11 through 21.

11.1 TRANSCEIVER SETUP RELATIONSHIPS.

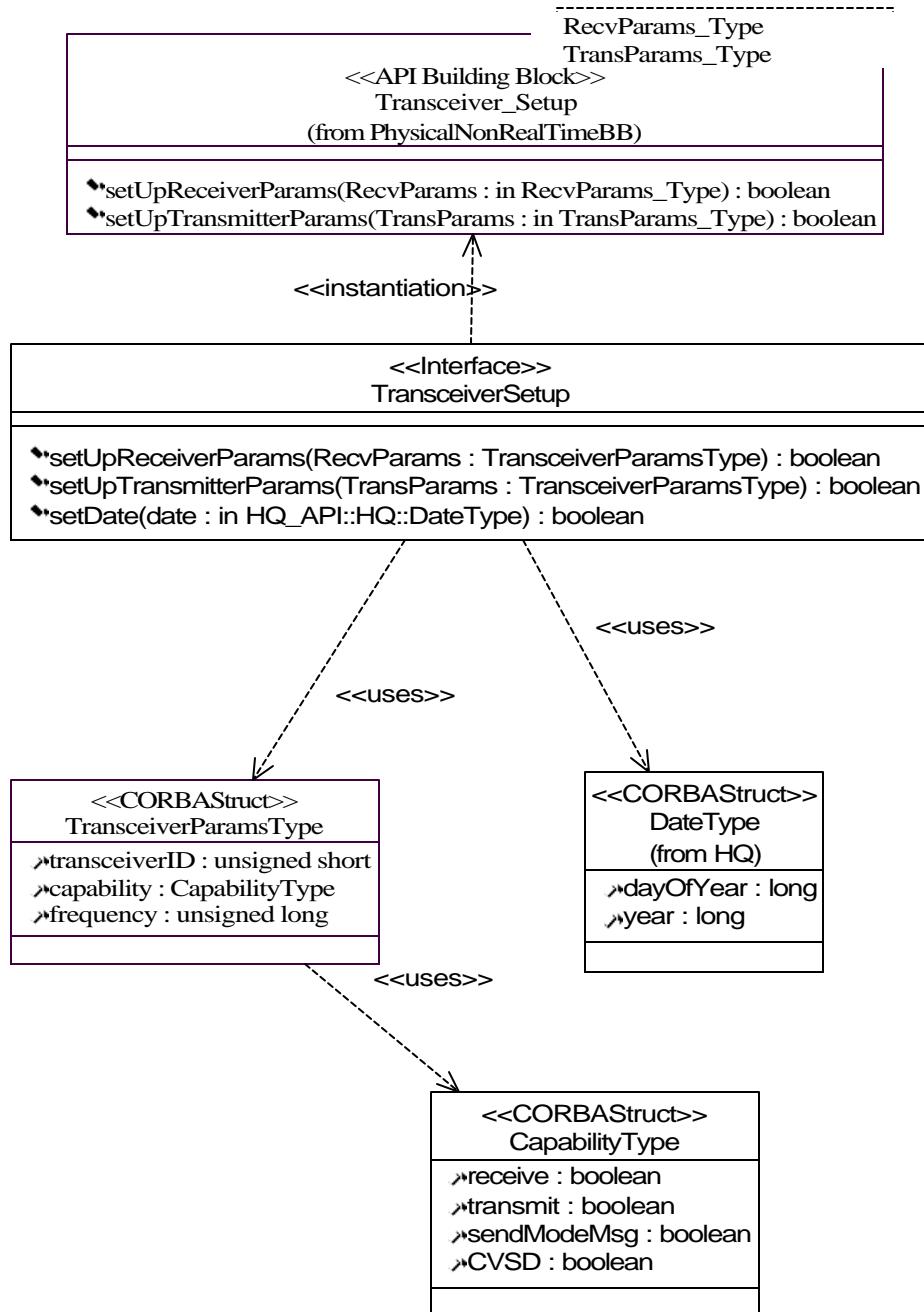


Figure 11. Transceiver Setup Relationships

11.2 HAVE QUICK RADIO MODE RELATIONSHIPS.

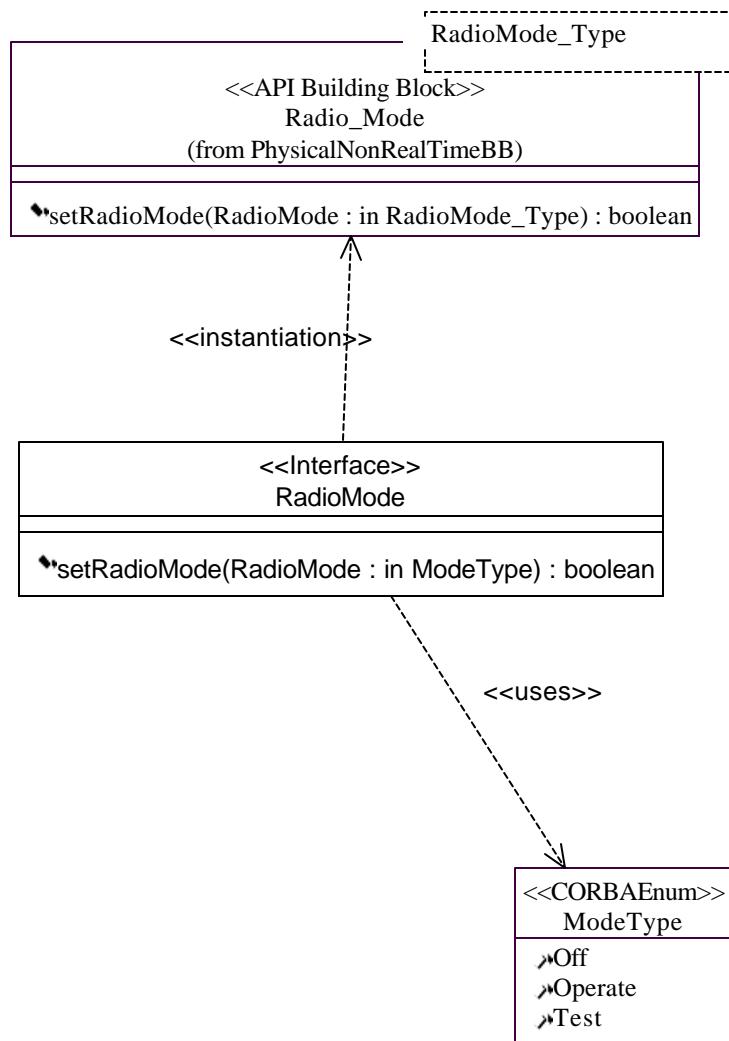


Figure 12. Have Quick Radio Mode Relationships

11.3 HAVE QUICK MEDIA SETUP RELATIONSHIPS.

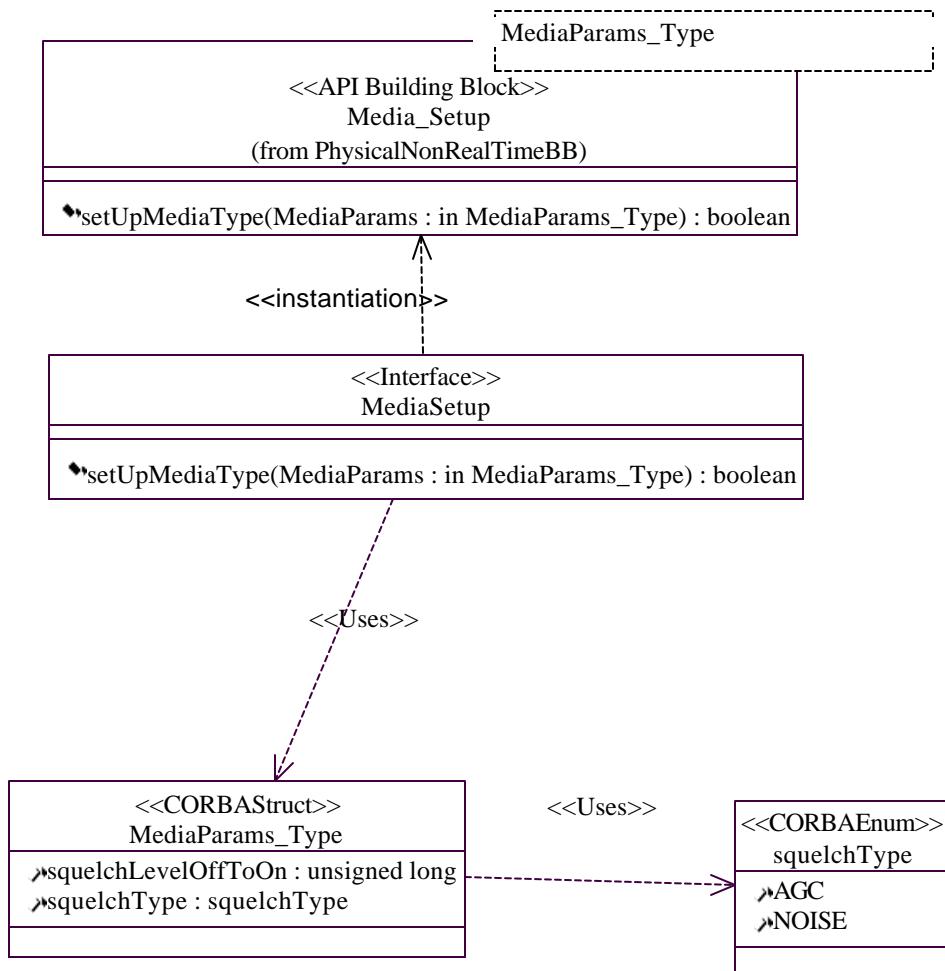


Figure 13. Have Quick Media Setup Relationships

11.4 HAVE QUICK RECEIVE COMMAND RELATIONSHIPS.

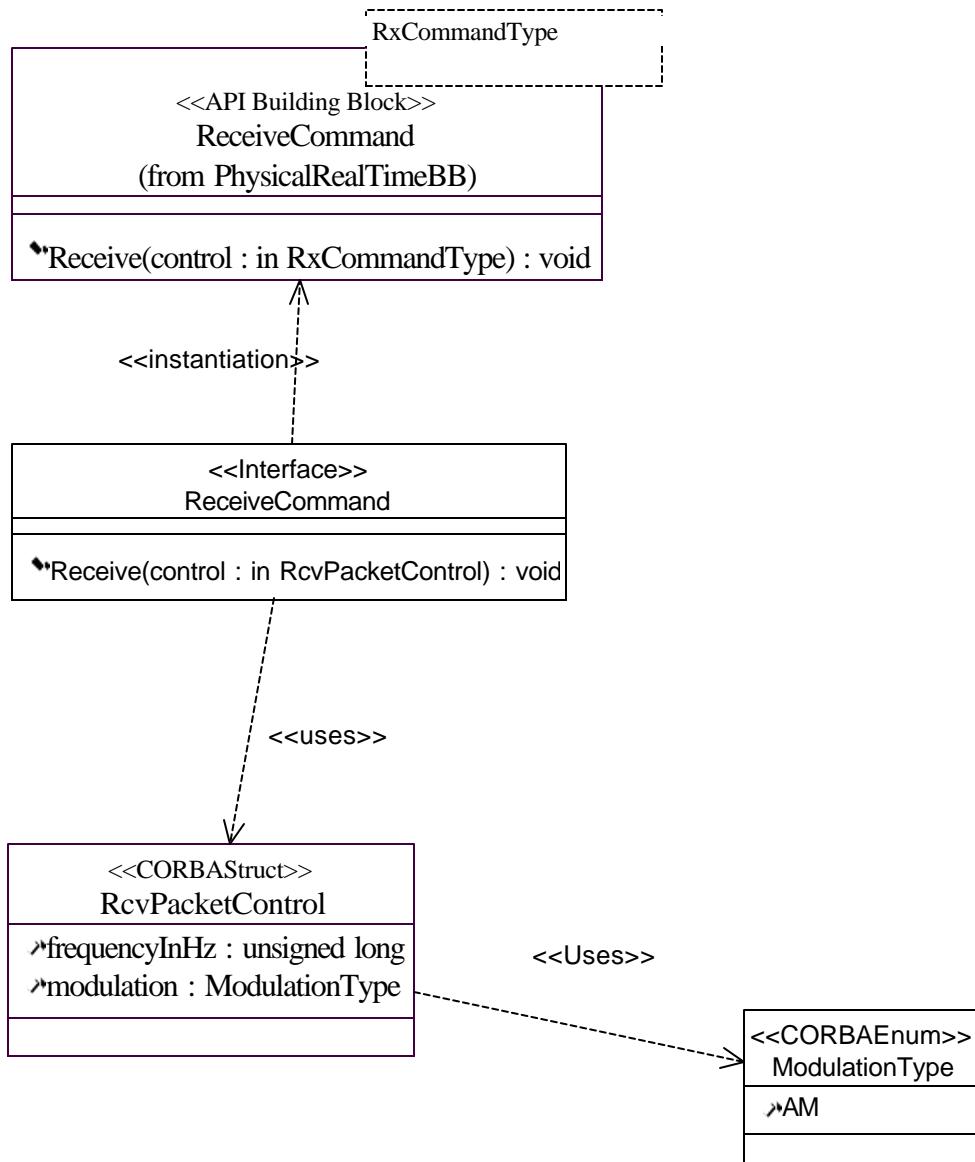
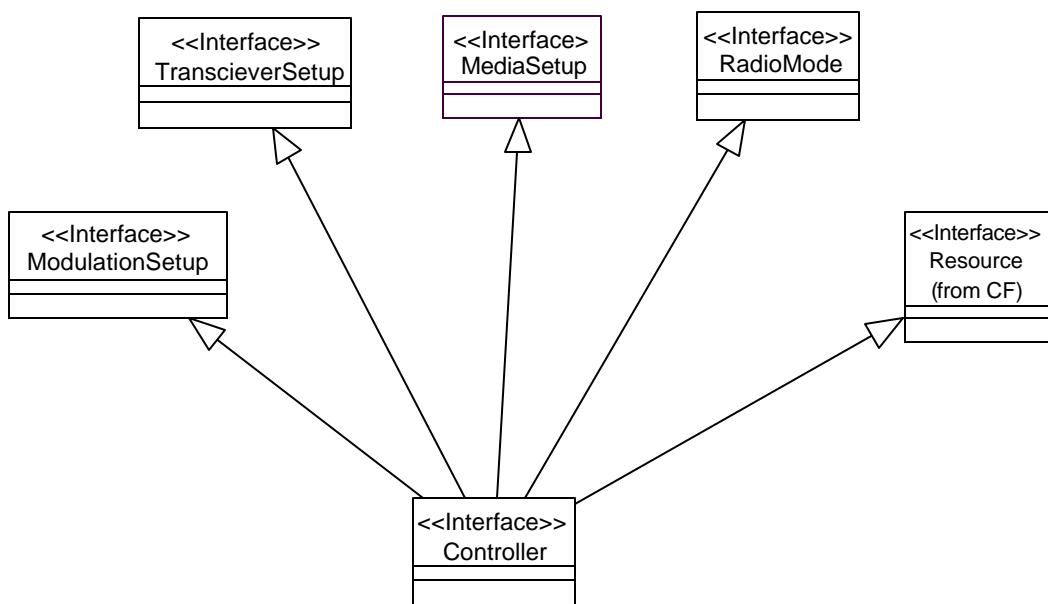


Figure 14. Have Quick Receive Command Relationships

11.5 NON-REAL-TIME SERVICE PROVIDER INHERITANCE.**Figure 15. Non-Real-Time Service Provider Inheritance**

11.6 HAVE QUICK ULONGPACKET RELATIONSHIPS.

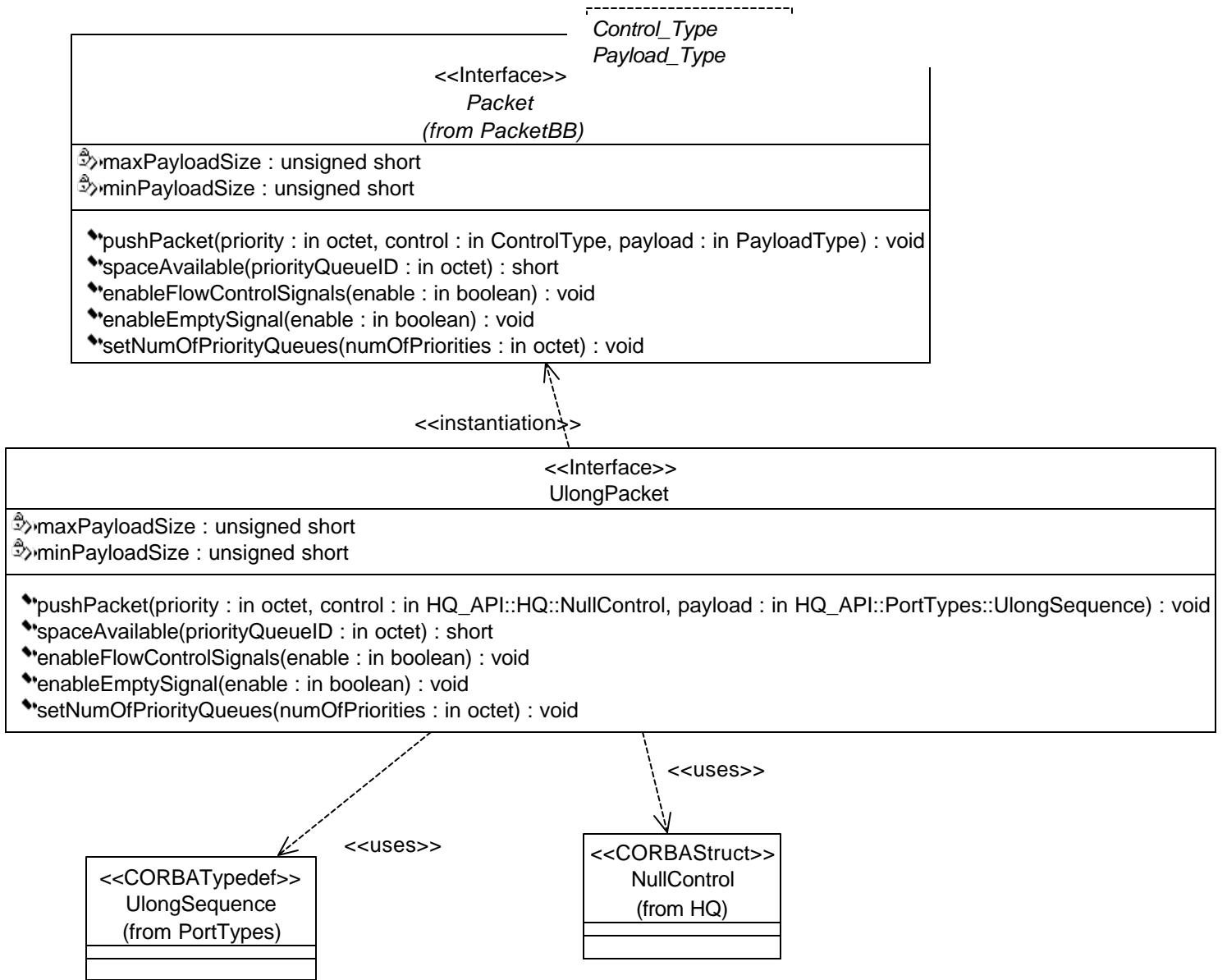


Figure 16. Have Quick UlongPacket Relationships

11.7 HAVE QUICK USHORTPACKET RELATIONSHIPS.

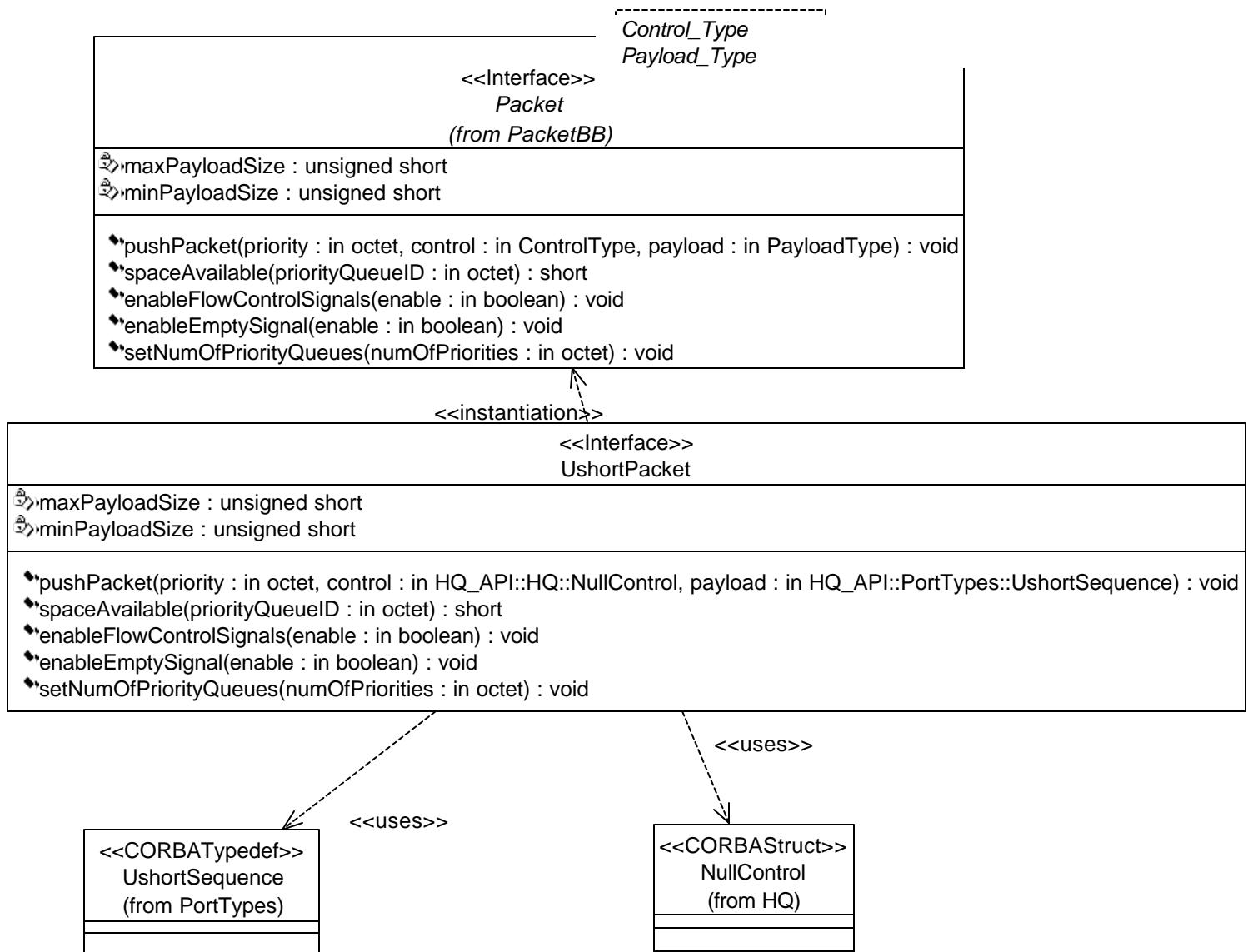


Figure 17. Have Quick UshortPacket Relationships

11.8 HAVE QUICK OCTETPACKET RELATIONSHIPS.

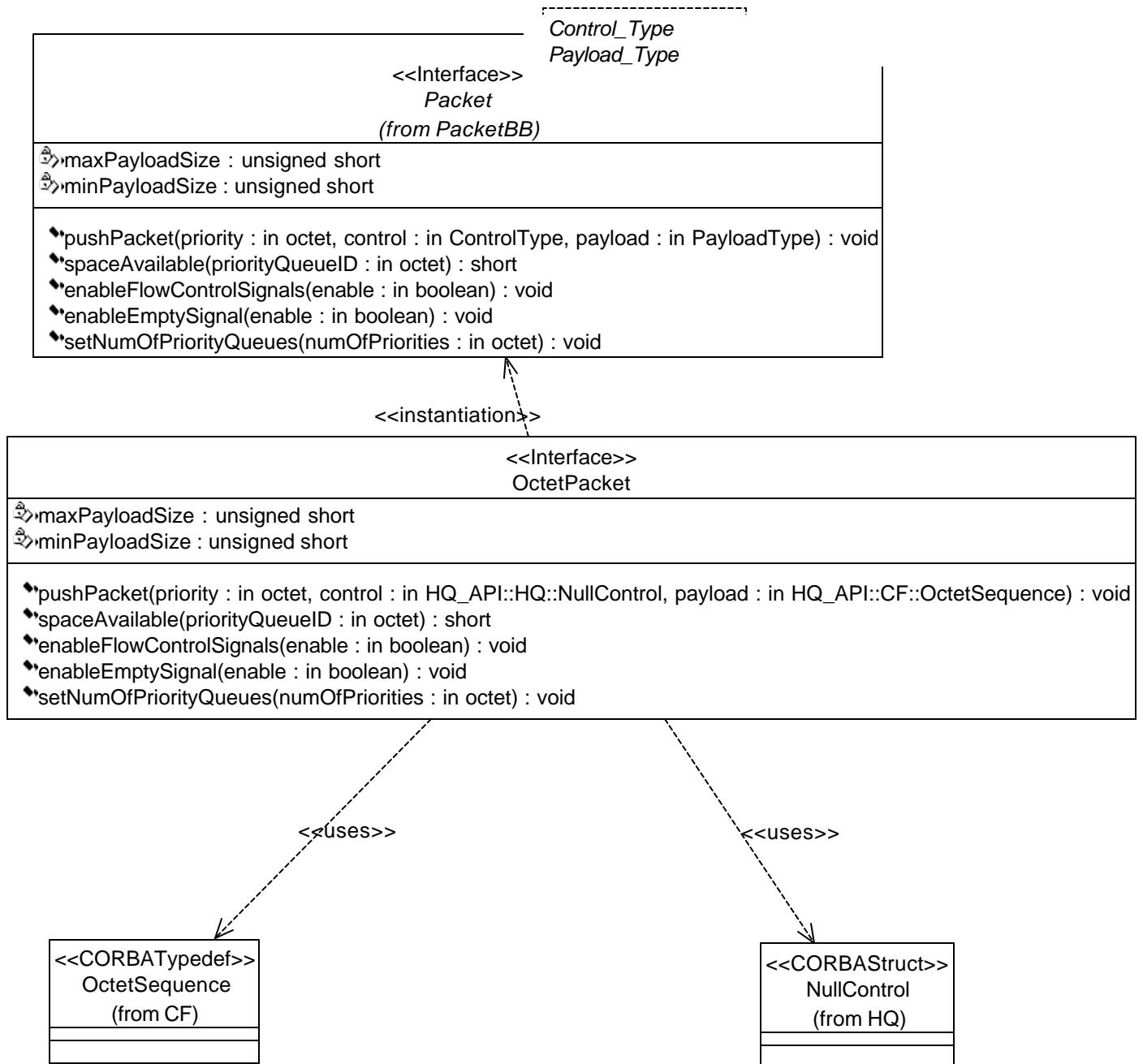


Figure 18. Have Quick OctetPacket Relationships

11.9 REAL-TIME SERVICE PROVIDER INHERITANCE.

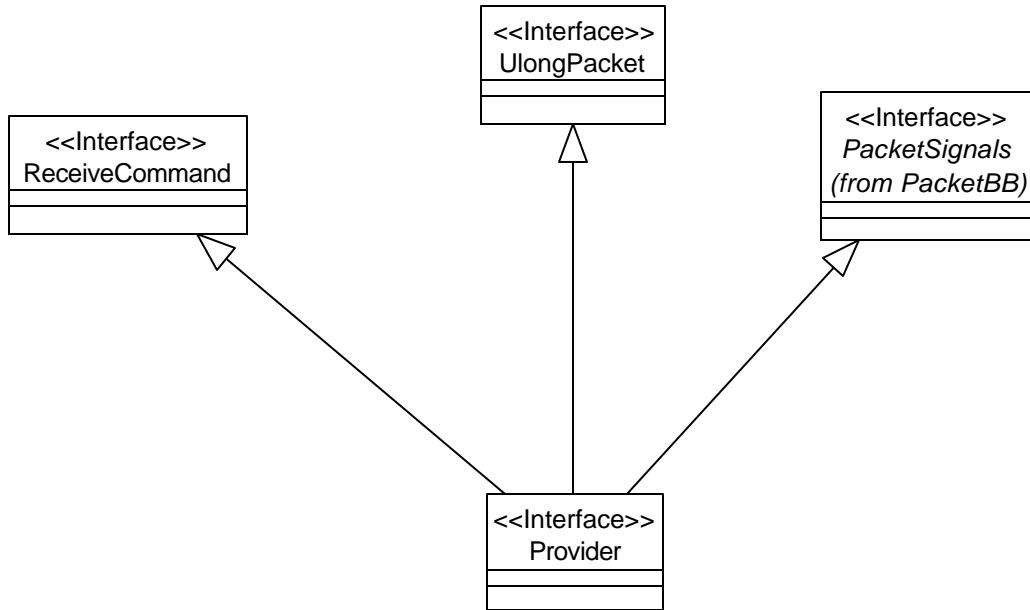


Figure 19. Real-Time Service Provider Inheritance

11.10 REAL-TIME SERVICE USER INHERITANCE.

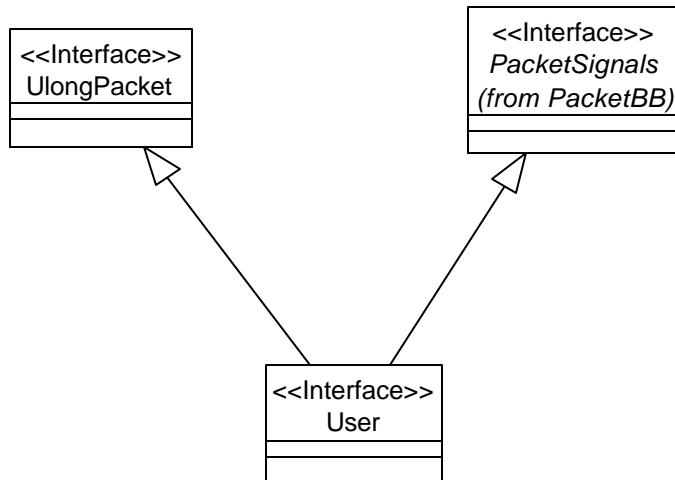
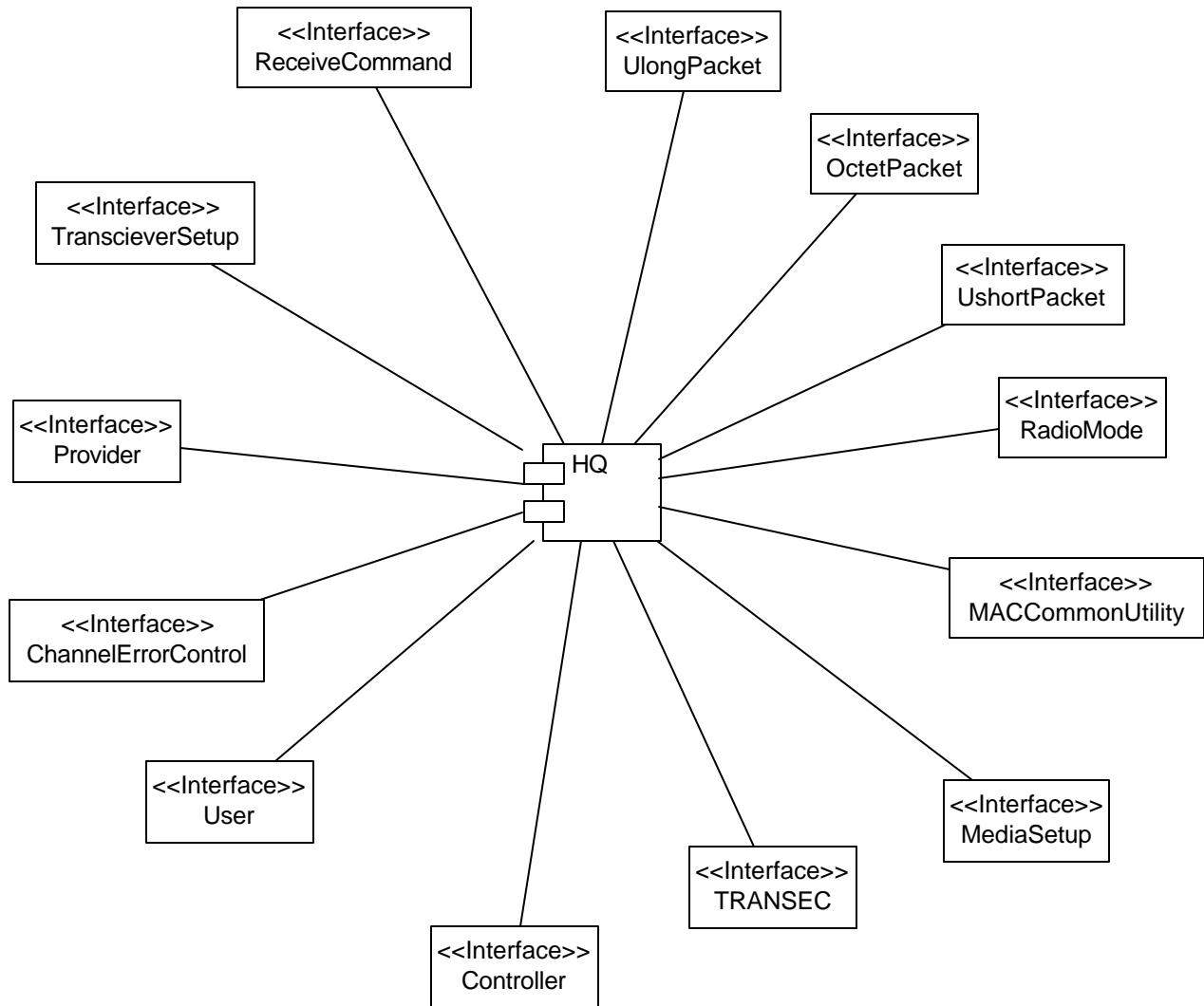


Figure 20. Real-Time Service User Inheritance

11.11 HAVE QUICK COMPONENT DIAGRAM.**Figure 21. Have Quick Component Diagram**